ECE 3663 - Digital Integrated Circuits
University of Virginia

Design Review I

Report Submitted by:
Team Awesome
Mark Cheung, Michael Recachinas, Piyapath Siratarnsophon, David Yi

On our honor, we have neither given nor received unauthorized assistance on this project.

April 2, 2013

Design Review I

Abstract

The project is to design, simulate, and build an arithmetic logic unit (ALU) contained in a digital signal processor (DSP) using CMOS transistors and Cadence Free PDK design kit. Within the ALU will be basic logic functions (AND, OR, NOT) as well as more complicated functions such as add and subtract. The functions within the ALU are to effectively be chosen by an 8-to-1 multiplexer (MUX). Using latches, ultimately registers will be implemented to store the values on the rising edge of the clock. The goal of the project is to minimize the power dissipation, latency, and overall area of the ALU, while ensuring accurate and precise output.

1 Design Approach

For our initial design of AND, OR, PASS A, and the multiplexer, we chose successful designs, perhaps with longer delay, over optimized designs. For PASS A, the design was simple, just have a wire of the correct bit width from the input to the output. For AND and OR, we built CMOS NAND and NOR gates and then inverted their respective outputs. Although this cascade contains delay, the overall ALU delay will be so much greater that this delay does not make a difference. For the multiplexer, we built a simple 2:1 MUX using an inverter design (figure 8), and then connected 7 2:1 MUXes to build an 8:1 MUX.

2 Schematics

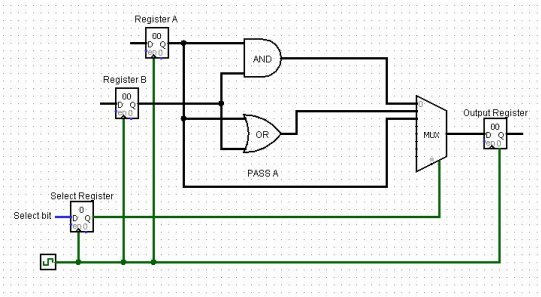

Figure 1: Initial Block diagram of ALU Block in DSP with an 8:1 MUX and basic functions (AND, OR, PASS A) (from *Logisim*)

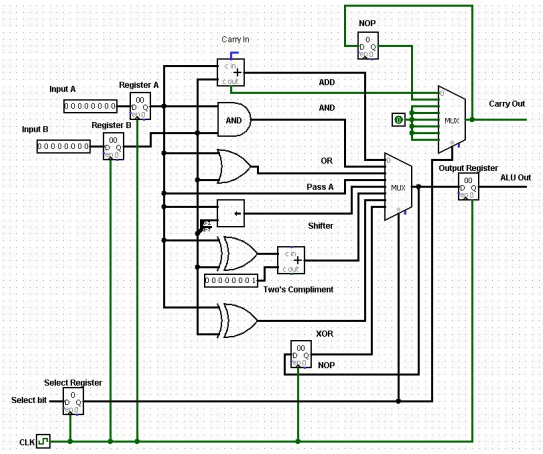

Figure 2: Initial Block diagram of ALU Block in DSP with all required functions (from *Logisim*)

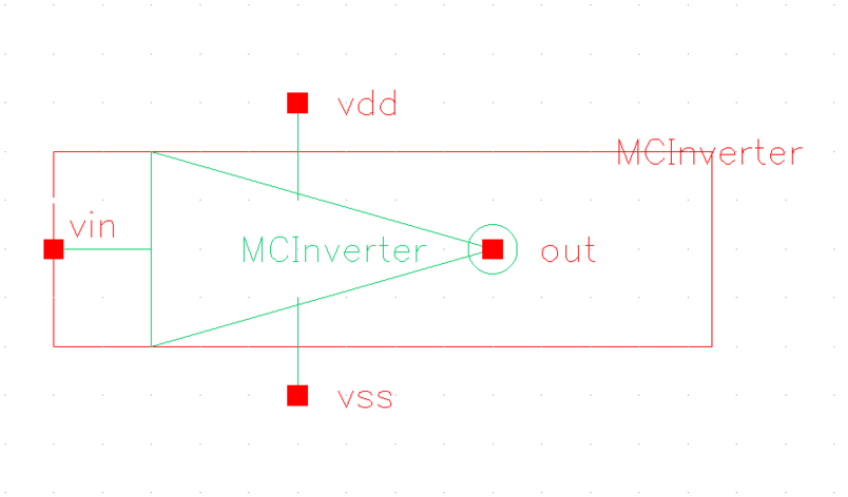

Figure 3: Inverter Symbol (from *Cadence*)

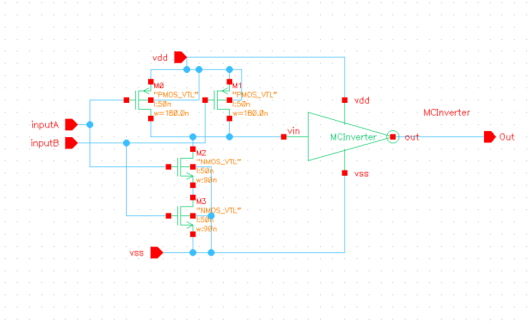

Figure 4: AND gate design (from *Cadence*)

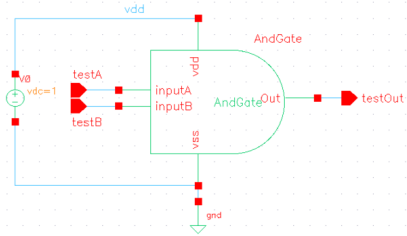

Figure 5: AND gate symbol (from *Cadence*)

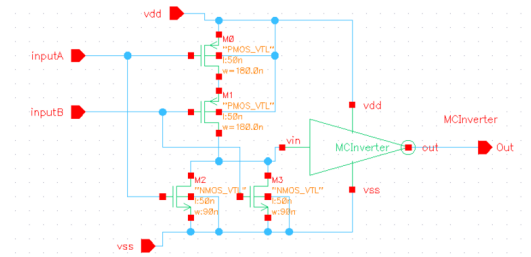

Figure 6: OR gate design (from *Cadence*)

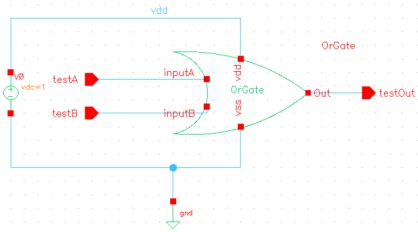

Figure 7: OR gate symbol (from *Cadence*)

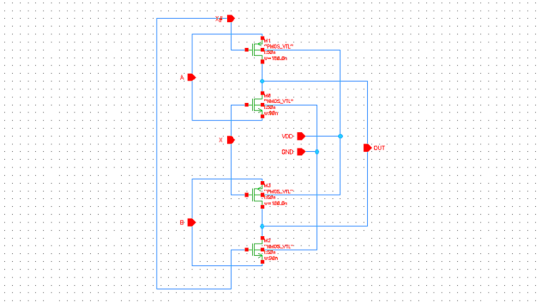

Figure 8: 2-to-1 MUX design (from *Cadence*)

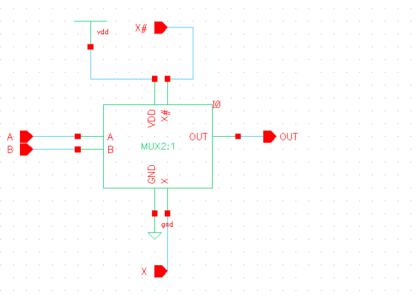

Figure 9: 2-to-1 MUX symbol (from *Cadence*)

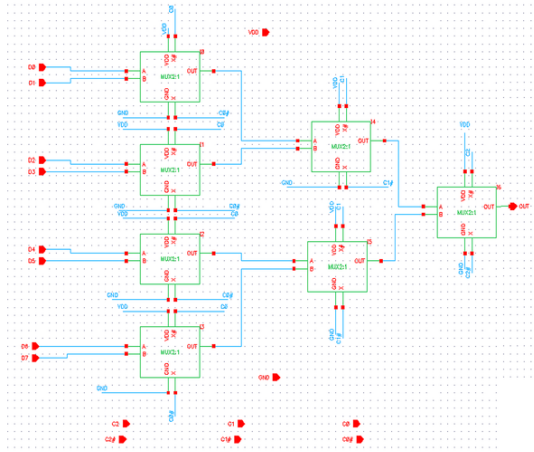

Figure 10: 8-to-1 MUX design (from *Cadence*)

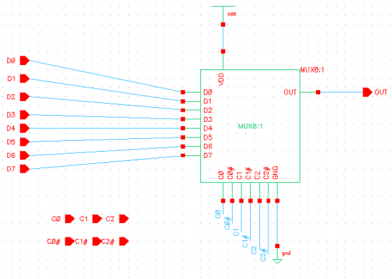

Figure 11: 8-to-1 MUX symbol (from *Cadence*)

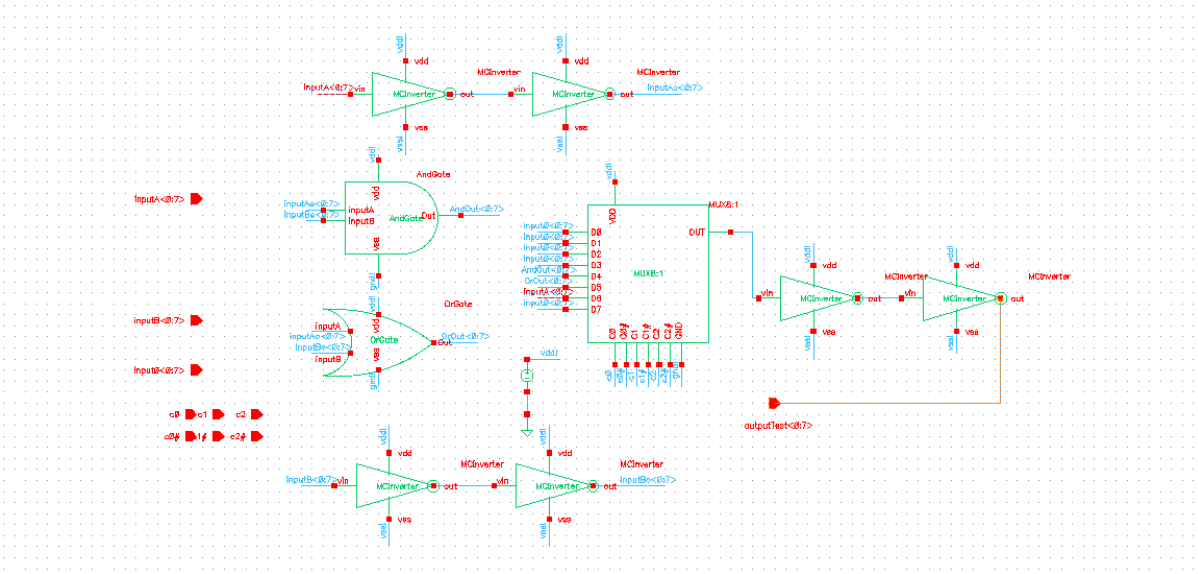

Figure 12: Everything incorporated into one schematic (from *Cadence*)

3 Test Strategy

To test the built components, we plan to simulate each input by injecting direct current (DC) input as well as pulses.

Our test strategy is as follows:

1. "Check and save" the component to ensure it is error and warning free.
2. Launch the ADE L simulation.
3. Load modules under their respective model library. (Primarily for NMOS and PMOS transistors.)
4. Configure setup analog stimuli or the inputs.
5. Setup for transient testing with the start and end time input.
6. Run the simulation for a netlist output log to show any possible errors or warnings in the component with input stimuli.
7. If needed, the netlist output log can be used to run simulations on Ocean.
8. Plot the output (Results → Direct Plot → Transient Signal.
9. Compare simulated outputs with expected and theoretical results to diagnose any possible discrepancies.
10. The previous steps can be used individually for AND, OR, PASS A, and MUX, as well as all of the functions combined.

For the AND and OR gates, we plan to use all of the 4 possible input states and check that the output works properly (according to the truth tables below).

AND

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

OR

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

PASS A can be tested by checking that the signal from A propagates successfully to the output.

For the multiplexers, we plan to test the individual inputs with their respective select bits chosen and match the input to the output on the graphs.

When we are satisfied with each individual block component, we will test the functionality as a whole system.

4 Simulation Results

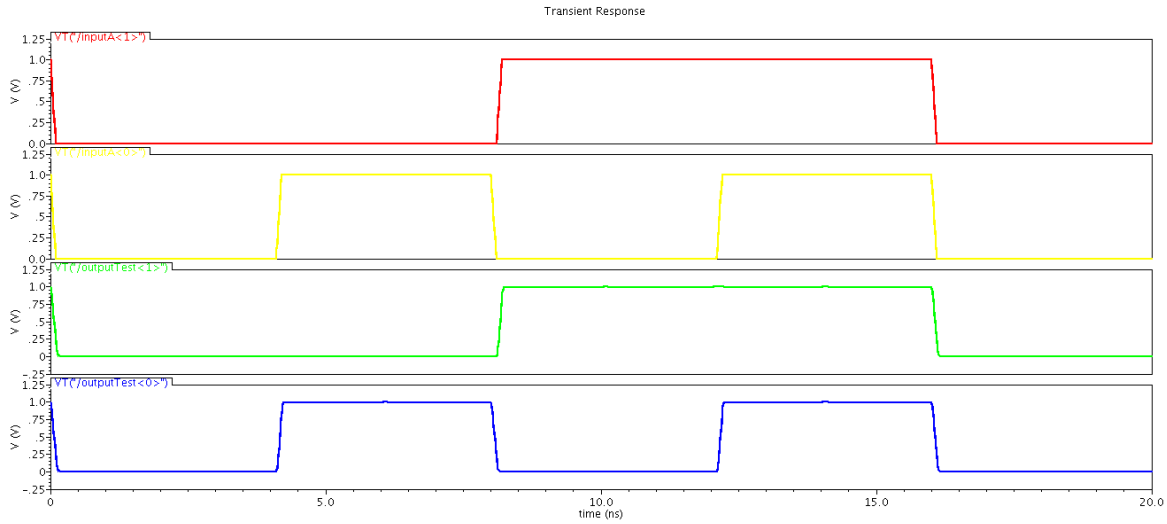

Figure 13: PASS A simulation (from *Cadence*)

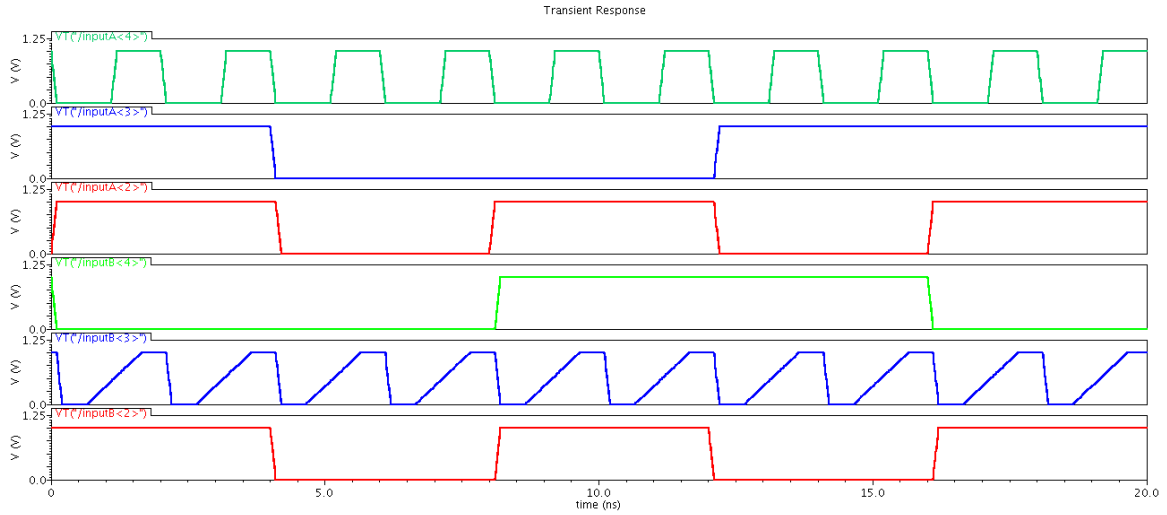

Figure 14: AND input simulation (from *Cadence*)

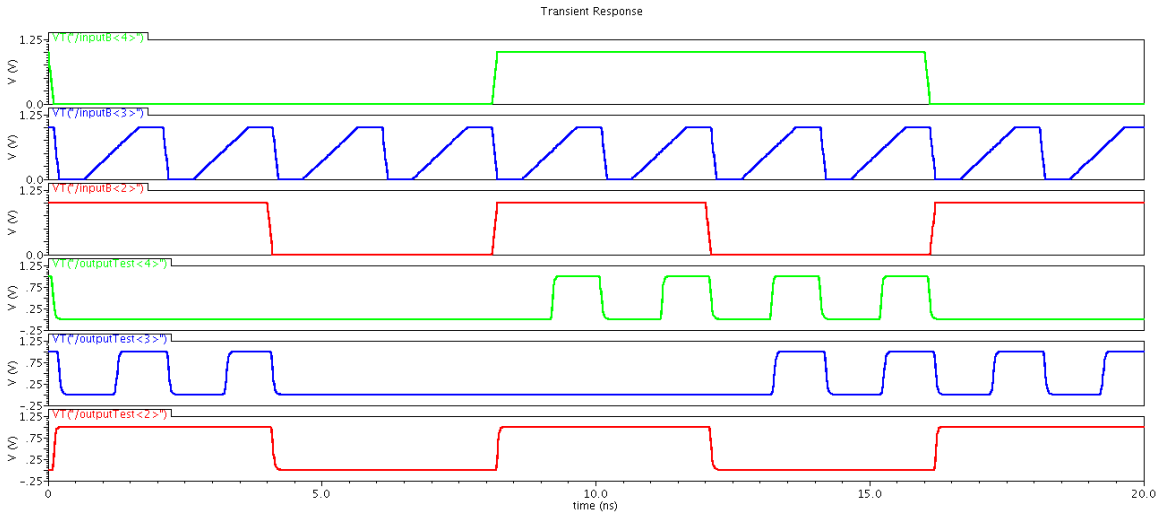

Figure 15: AND output simulation (from *Cadence*)

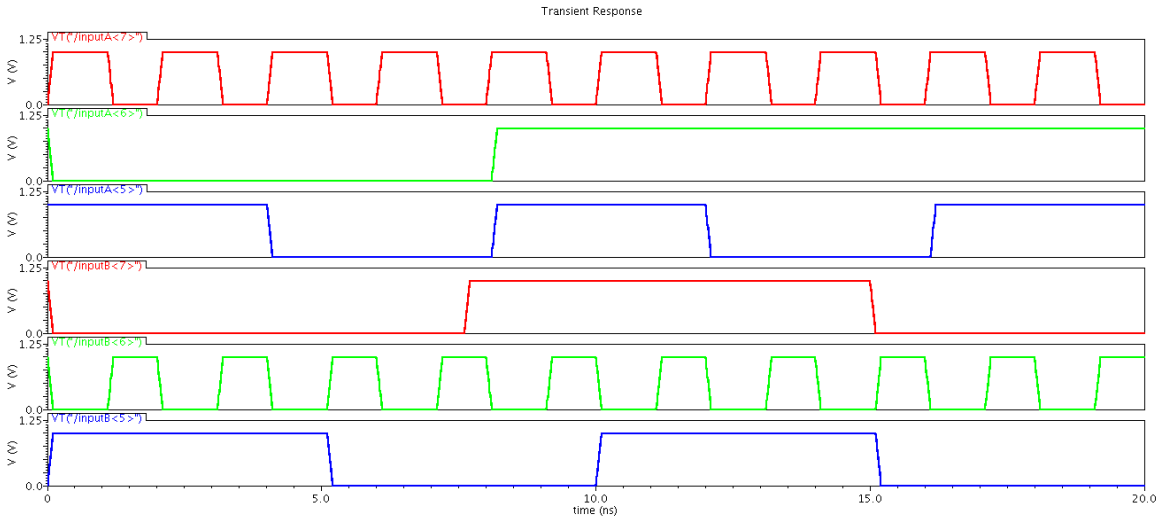

Figure 16: OR input simulation (from *Cadence*)

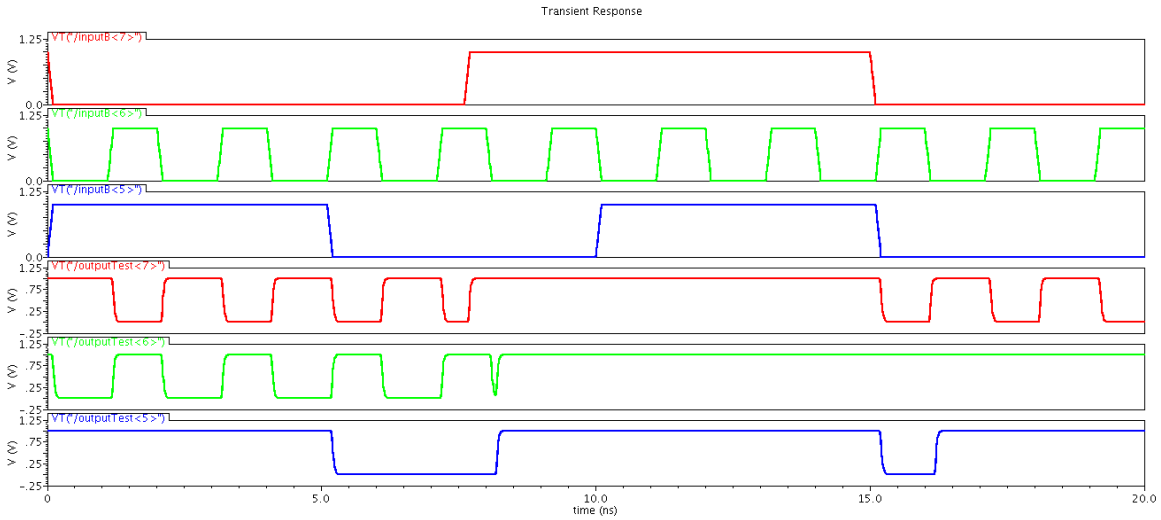

Figure 17: OR output simulation (from *Cadence*)

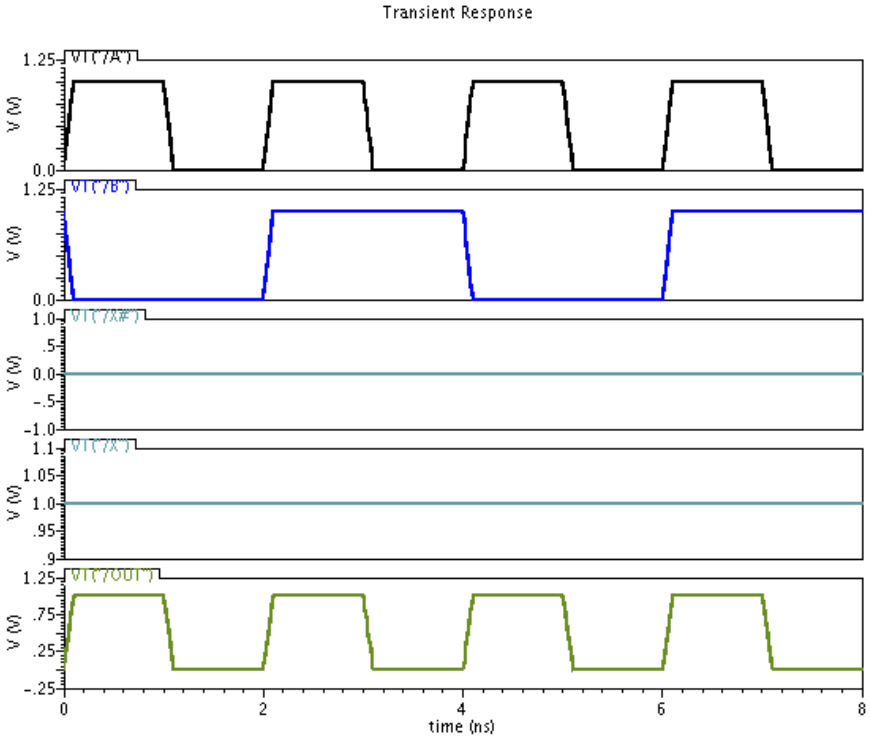

Figure 18: 2:1 MUX simulation (from *Cadence*)

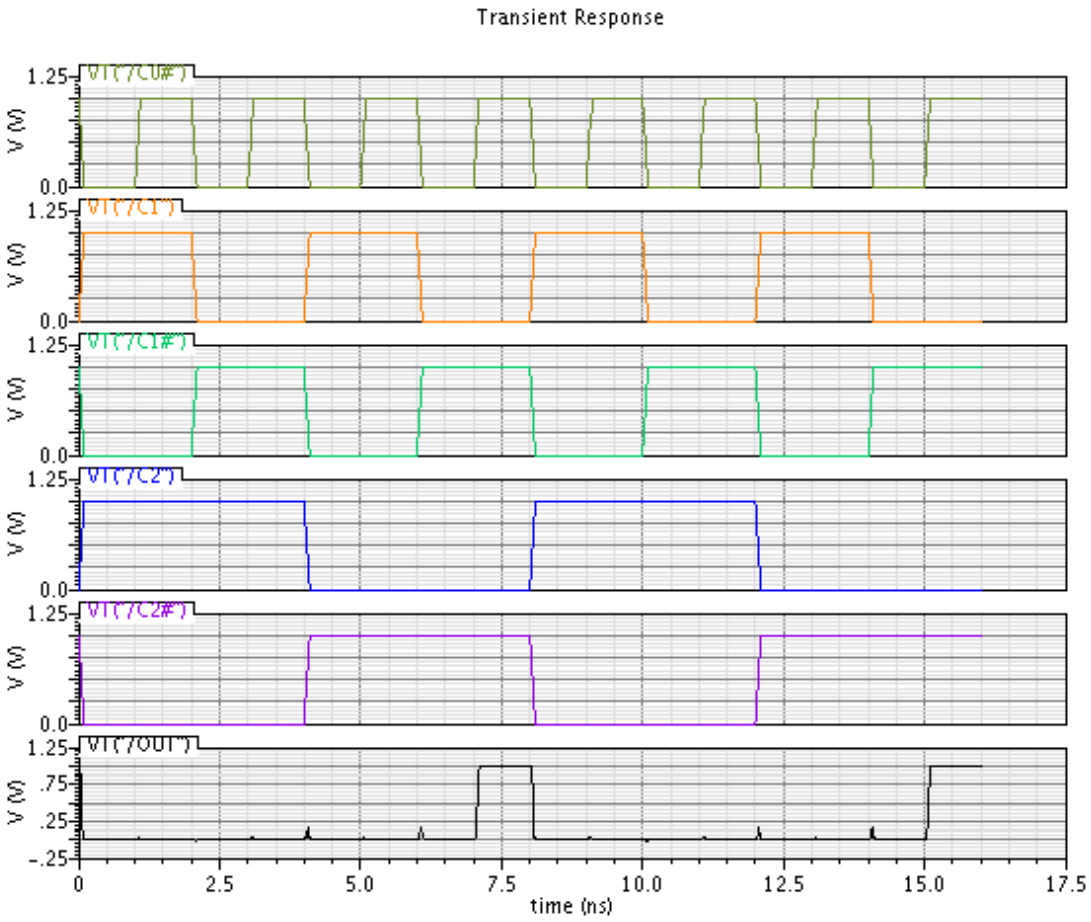

Figure 19: 8:1 MUX simulation (from *Cadence*)

5 Work Breakdown Schedule

5.1 Current Group Progress

We have completed the initial designs of the ALU diagram and schematics for the four components: AND, OR, PASS A, and 8:1 MUX. The schematics have been fully tested and verified using Ocean. Schematic design, test documentation, plan, and logged meetings can be found on our Wiki:

`https://venividiwiki.ee.virginia.edu/mediawiki/index.php/ClassECE3663Spring13/ClassECE3663Spring13ProjectAwesome`

5.2 Plan for Design Review II

→ By April 6, 2013: ADD, 2 COMP, XOR and SHIFT schematics should be done by then. On Saturday 6, the group will meet in Rice 414 to test the components and perform debugging together.

→ By April 7, 2013: All components should be tested by Sunday, but if they are not, the group will finish debugging the functions by this date. The components will be put together into one ALU by the whole group. A design review 2 report will be put together at the end. When the above is done, the group will make a presentation and decide on roles.

→ By April 9, 2013: The members will meet together to rehearse and review their presentation. By Tuesday, the DSP system should be fully functional with enough tests completed successfully to verify proper functionality.

5.3 Plan for Final Design

→ By April 13, 2013: The group will work to optimize the system.

→ By April 20, 2013: By this date, the system should be fully optimized, debugged, and functioning well. The group will meet together to create a presentation of the final design and a final report.

5.4 Arbitrary Function Ideas

Currently we have two main ideas for an arbitrary function: an exclusive-or (XOR) or a Wallace Tree Multiplier. The XOR would aid in making the adder, while also acting as a separate function. The XOR could be used as a cipher for encryption purposes (where input A is the message and input B is the key). Due to the overlap between the XOR and the adder, our group has contemplated implementing a Wallace Tree Multiplier. After some research, it appears the Wallace Tree Multiplier has a much shorter delay than other multipliers (e.g. binary, etc.) and it's area is much smaller than its alternatives. If we pursue the multiplier, it would certainly be more of a challenge than the XOR, but it would surely add more functionality to the final system.